LOVELY PROFESSIONAL UNIVERSITY

CSE211: COMPUTER ORGANIZATION AND DESIGN
Course Assessment Model

Lectures Per week: 3, Tutorials per week: 1

Class Attendance	:	5 marks
2 Class Assessments and 1 Quiz	:	20 marks
Mid Term Exam	:	25 marks
End Term Exam	:	50 marks
Total Marks	:	**100 marks**

LOVELY PROFESSIONAL UNIVERSITY

CSE211: COMPUTER ORGANIZATION AND DESIGN
Unit I

Introduction to digital electronics : logic gate, introduction to combinational circuits, introduction to sequential circuits

Introduction to basics of architecture : computer architecture, computer organization

Register Transfer and Micro Operations : Bus and Memory Transfer, Logic Micro operations, Shift Micro Operations, Register transfer and register transfer language, Design of arithmetic logic unit., arithmetic micro operations, arithmetic logic shift unit

LOVELY PROFESSIONAL UNIVERSITY

CSE211: COMPUTER ORGANIZATION AND DESIGN
Objective of This Subject in Course

- To understand the generic principles that underlie the building of a digital computer.
- To review the structure and functioning of a digital computer and understand its overall system architecture.
- To analyze the working of memory unit and study the examples of mapping techniques of different memory systems.

LOVELY PROFESSIONAL UNIVERSITY

Introduction to Digital Electronics

- Digital Systems are used in communication, business transactions, traffic control, space guidance, medical treatments, weather monitoring, the internet and many other commercial ,industrial and scientific enterprises.
- Digital System can manipulates a discreet element of information, such discreet elements of information can be electric impulses, the decimal digits, the letter of an alphabets, arithmetic operations or any other set of meaningful symbols.
- Discreet quantity of information arises either form the nature of the process or may be quantized from a continuous process.

LOVELY PROFESSIONAL UNIVERSITY

CSE211: COMPUTER ORGANIZATION AND DESIGN
Books Preferred

- By M. Morris Mano **"COMPUTER SYSTEM ARCHITECTURE"**, Published by PEARSON Publication
- By HENNESSY,J.L,DAVID A PATTERSON, AND GOLDBERG " **COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH"** Published by PEARSON Publication
- By P. PAL CHOUDHURI "**COMPUTER ORGANISATION AND DESIGN"** Published by PRENTICE HALL

LOVELY PROFESSIONAL UNIVERSITY

Basic Elements of Digital Electronics

- Logic Gates
- Decoder
- Encoder
- Multiplexer
- De-multiplexer
- Flip-flop

Logic Gates

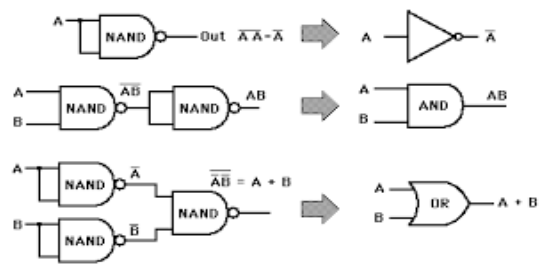
Basic Logic Gates

- AND Gate
- OR Gate
- NOT Gate

Universal Logic Gates

- NAND Gate
- NOR Gate

Basic Gates using NAND Gate



Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol	A, X	A, B, X					

Truth Table

A	X
0	1
1	0

B	A	X
0	0	0
0	1	0
1	0	0
1	1	1

B	A	X
0	0	1
0	1	1
1	0	1
1	1	0

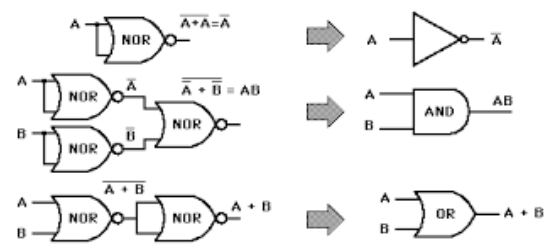
B	A	X
0	0	0
0	1	1
1	0	1
1	1	1

B	A	X
0	0	1
0	1	0
1	0	0
1	1	0

B	A	X
0	0	0
0	1	1
1	0	1
1	1	0

B	A	X
0	0	1
0	1	0
1	0	0
1	1	1

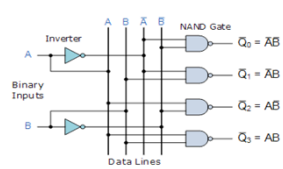
Basic Gates using NOR Gate



Universal Logic Gates

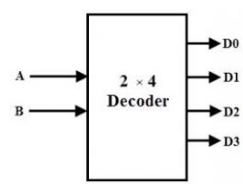
- A universal gate is a gate which can implement any Boolean function without need to use any other gate type.
- The NAND and NOR gates are universal gates.
- In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families

Decoder

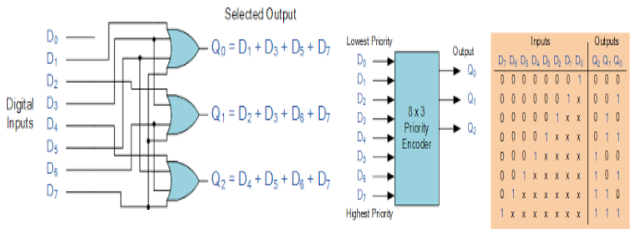


Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0



Encoder (Priority Encoder)



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = constant

Applications of Multiplexers

- Computer Memory

  Multiplexers are used in computer memory to maintain a huge amount of memory in the computers, and also to reduce the number of copper lines required to connect the memory to other parts of the computer.

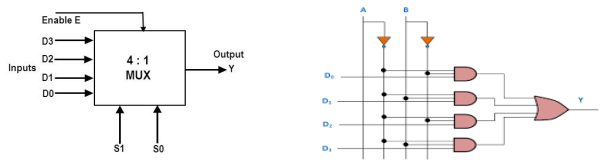
- Telephone Network

  In telephone networks, multiple audio signals are integrated on a single line of transmission with the help of a multiplexer.

- Transmission from the Computer System of a Satellite

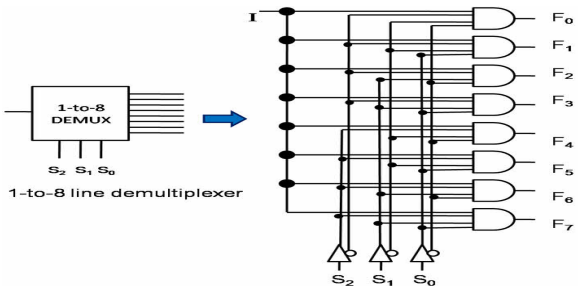
  Multiplexer is used to transmit the data signals from the computer system of a spacecraft or a satellite to the ground system by using a GSM satellite.

Multiplexers



- In the large-scale-digital systems, a single line is required to carry on two or more digital signals – and, of course! At a time, one signal can be placed on the one line. But, what is required is a device that will allow us to select; and, the signal we wish to place on a common line, such a circuit is referred to as multiplexer.

De-Multiplexers



Applications of Multiplexers

Multiplexers are used in various applications wherein multiple-data need to be transmitted by using single line.

- Communication System

  A communication system has both a communication network and a transmission system. By using a multiplexer, the efficiency of the communication system can be increased by allowing the transmission of data, such as audio and video data from different channels through single lines or cables.

Applications of De multiplexer

De multiplexers are used to connect a single source to multiple destinations. These applications include the following:

- Communication System

  Mux and de-mux both are used in communication system to carry out the process of data transmission. A De-multiplexer receives the output signals from the multiplexer and at the receiver end it converts them back to the original form.

- Arithmetic Logic Unit

  The output of the ALU is fed as an input to the De-multiplexer, and the output of the de-multiplexer is connected to a multiple register. The output of the ALU can be stored in multiple registers.

Applications of De multiplexer

- Serial to Parallel Converter

This converter is used to reconstruct parallel data. In this technique, serial data is given as an input to the De-multiplexer at a regular interval, and a counter is attached to the de-multiplexer at the control input to detect the data signal at the output of the de-multiplexer. When all data signals are stored, the output of the de-mux can be read out in parallel.

Sequential Circuits

Latches

- NAND Latch
- NOR Latch

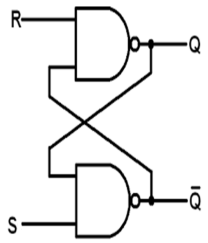
Flip-Flops

- S-R Flip Flop (Set-Reset)
- D Flip Flop
- J-K Flip Flop (Jack-King)
- T Flip Flop

Combinational Circuits

- No feedback paths
- No memory
- Combinational circuit is a connected arrangement of logic gates with set of inputs and outputs.
- Binary values of outputs are a function of binary combination of inputs.

NAND Latch

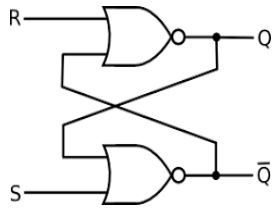


S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Sequential Circuits

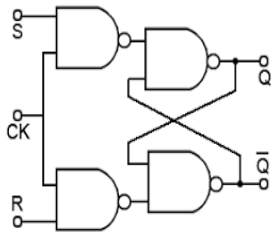
- Feedback paths exist
- Memory present
- 2 Types- Synchronous and Asynchronous
- Synchronous seq circuits employ signals that effect storage elements only at discrete instants of time.
- Synchronization is achieved with help of device called clock

NOR Latch



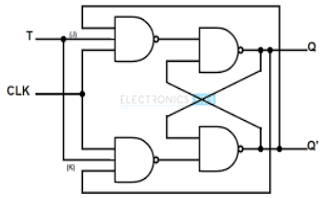
S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

SR Flip Flop



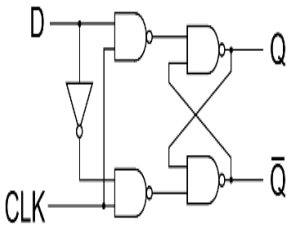
CLK	S	R	Q_n	Q_{n+1}
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	X
1	1	1	1	X

T Flip Flop



CLK	T	Q_n	Q_{n+1}
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

D Flip Flop



CLK	D	Q_n	Q_{n+1}
0	X	0	0
0	X	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Excitation Tables of FFs

SR Flip-flop

Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

D Flip-flop

Q(t)	Q(t+1)	DR
0	0	0
0	1	1
1	0	0
1	1	1

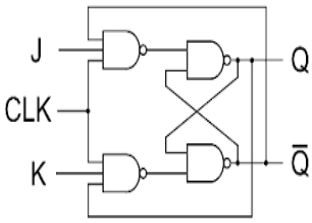
JK flip-flop

Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

T flip-flop

Q(t)	Q(t+1)	DR
0	0	0
0	1	1
1	0	1
1	1	0

JK Flip Flop



CLK	J	K	Q_n	Q_{n+1}
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Introduction to basics of Architecture

Computer Organization and Architecture

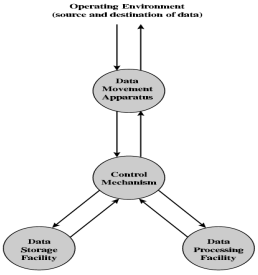
Computer Organization - the internal arrangements of a computer, which includes the design of the processor, memory and input/ output circuits.

Computer Architecture describes features of a computer family (notably the instructions) and not the specific implementation, just like architecture of a house might be described as Victorian.

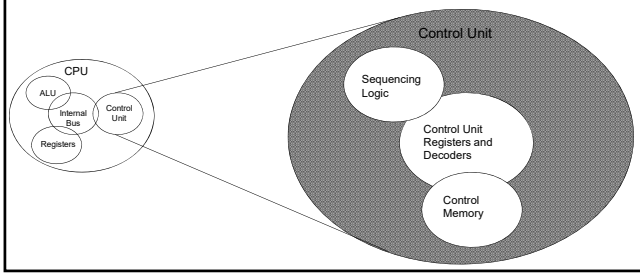
Some do not differentiate between ***computer organization*** and ***computer architecture***.

Introduction to basics of Architecture

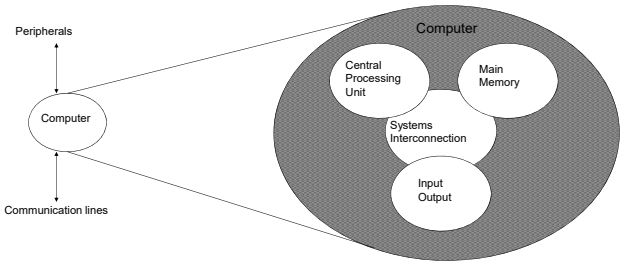
- All computer functions are:
  - Data processing
  - Data storage
  - Data movement
  - Control mechanism



System Structure - The Control Unit



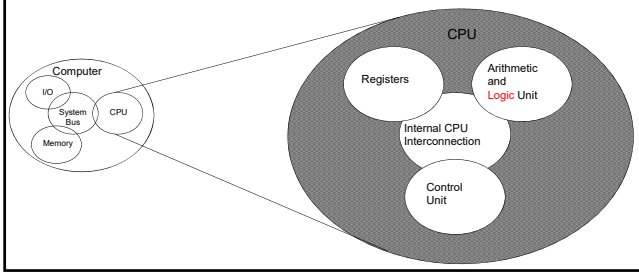
System Structure - Top Level



Bus Organization

- The CPU communicates with the other components via a bus. A *bus* is a set of Wires (multiplexers) that acts as a shared but common data path to connect multiple subsystems within the system. It consists of multiple lines, allowing the parallel movement of bits.
- Buses are low cost but very versatile, and they make it easy to connect new devices to each other and to the system. At any one time, only one device (be it a register, the ALU, memory, or some other component) may use the bus. However, this sharing often results in a communications bottleneck. The speed of the bus is affected by its length as well as by the number of devices sharing it.

System Structure - The CPU

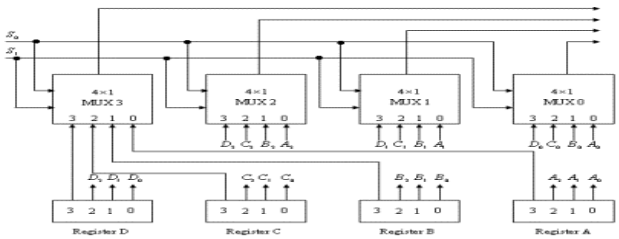


Common Bus configuration System

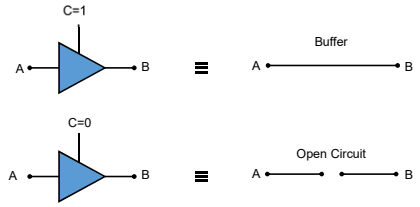
A more efficient scheme for transferring information between common bus registers in a multiple-register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer. Constructing a common bus system

a. Using multiplexers
b. Using three state buffers.

Bus Configuration using Multiplexers



Three State Buffer



Bus Configuration using Multiplexers

The number of multiplexers needed to construct the bus is equal to n.

- The size of each multiplexer must be k x 1 since it multiplexes k data lines.

S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

For example: A common bus for eight registers of 16 bits each requires 16 multiplexers , one for each line in the bus. So Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

Construction of Bus by Three State Buffer

The outputs of four buffers are connected together to form a single bus line. (It must be realized that this type of connection cannot be done with gates that do not
have three-state outputs.) The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.

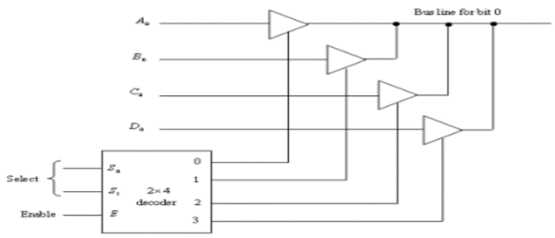
- No more than one buffer may be in the active state at any given time. The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high- impedance state.
- One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the diagram.

Bus Configuration using three State Buffer (Gate)

A three-state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate. The third state is a high impedance state.

- The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.
- Three-state gates may perform any conventional logic, such as AND or NAND. However, the one most commonly used in the design of a bus system is the buffer gate.

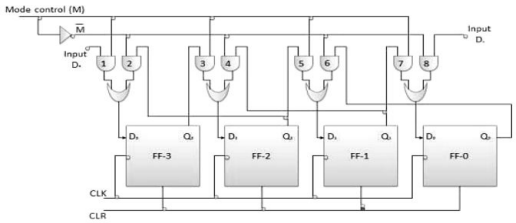
Bus Line with Three State Buffer



REGISTER

- A register is a group of flip-flops. Each flip–flop is capable of storing one bit of information. An n-bit register consists of a group of n flip-flops capable of storing n bits of binary information.
- In addition to the flip flops, a register may have combinational gates that perform certain data processing tasks. The flip-flops hold the binary information and the gates determine how the information is transferred into the register. Various types of registers are available commercially.
- The simplest register is one that consists of only flip flops without any gates.

Register with Combinational Circuits



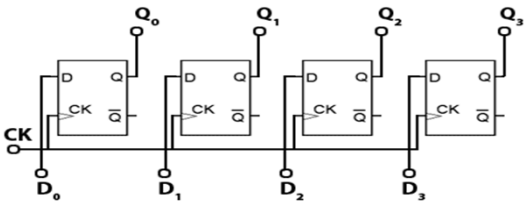
THE INTERNAL HARDWARE ORGANIZATION OF A DIGITAL COMPUTER IS BEST DEFINED BY SPECIFYING

1. The set of register it contains and their function.
2. The sequence of micro operations performed on the binary information stored in the registers.
3. The control that initiates the sequence of micro operations.

Register Transfer Language (RTL)

- The symbolic notation used to describe the micro operation transfers among register is called a register transfer language.
- A programming language is a procedure for writing symbols to specify a given computational process.
- A register transfer language is a system for expressing in symbolic form the micro operation sequences among the register of a digital module.

Register made by D Flip Flops



Register Transfer Language (RTL)

Basic Symbols for Register Transfers		
Symbol	Description	Examples
Letters & numerals	Denotes a register	MAR, R2
Parenthesis ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma ,	Separates two microoperations	R2 ← R1, R1 ← R2

Register Transfer Language (RTL)

- Information transfer from one register to another is designated in symbolic form by means of a replacement operator. The statement

  R2 ← R1

  denotes a transfer of the content of register R1 into register R2. It designates a replacement of the content of R2 by the content of R1.

- By definition, the content of the source register R1 does not change after the transfer. Normally, we want the transfer to occur only under a predetermined control condition. This can be shown by means of an if-then statement.

  If (P = 1) then (R2 ← R1)

  Where P is a control signal generated in the control section. It is sometimes convenient to separate the control variables from the register transfer operation by specifying a control function.

Register Transfer Language (RTL)

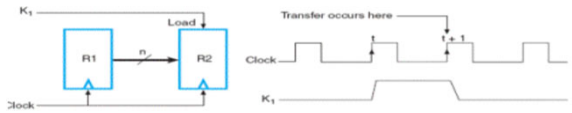
- A control function is a Boolean variable that is equal to I or 0. The control function is included in the statement as follows:

  P: R2 ← R1 The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if P= 1.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.

Register Transfer Language (RTL)

Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t + 1.this is shown in figure Below:

Transfer from R1 to R2 when K1=1



Register Transfer Language (RTL)

MEMORY TRANSFER

- A memory word will be symbolized by the letter M.
- The particular memory word among the many available is selected by the memory address during the transfer. This will be done by enclosing the address in square brackets following the letter M .Consider a memory unit that receives the address from a register, called the address register, symbolized by AR. The data are transferred to another register, called the data register, symbolized by DR Then:

Register Transfer Language (RTL)

- The register that holds an address for the memory unit is called a memory address register and is designated by the name MAR or AR. As for registers are PC (for program counter), IR (for instruction register) and R1 (for processor register).
- The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left.
- The most common way to represent a register is by a rectangular box with the name of the register inside .
- The name of the 16-bit register is PC. The symbol PC(O—7) or PC(L) refers to the low-order byte and PC(8—15) or PC(H) to the high-order byte.

Register Transfer Language (RTL)

Memory Read Operation: A read operation: the transfer of information from a memory word to the outside environment.

Read: DR ← M[AR]

This causes a transfer of information into DR from the memory word M selected by the address in AR. The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR.

Register Transfer Language (RTL)

Memory write operation: the transfer of new information to be stored into the memory.

Write: M[AR] ← R1

This causes a transfer of information from R1 into the memory word M selected by the address in AR.

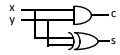
Half Adder & Full Adder

Half Adder

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

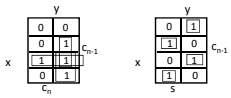
$c = xy$

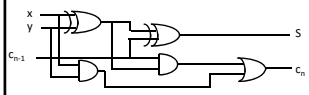
$s = xy' + x'y$
$= x \oplus y$



Full Adder

x	y	c_{n-1}	c_n	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1





$c_n = xy + xc_{n-1} + yc_{n-1}$
$= xy + (x \oplus y)c_{n-1}$

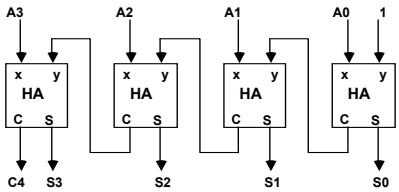
$s = x'y'c_{n-1} + x'yc'_{n-1} + xy'c'_{n-1} + xyc_{n-1}$
$= x \oplus y \oplus c_{n-1} = (x \oplus y) \oplus c_{n-1}$

Micro operations

Computer system micro operations are of four types:

- **Register transfer micro operations (RTL)**
- **Arithmetic micro operations**
- **Logic micro operations**
- **Shift micro operations**

Binary incrementer



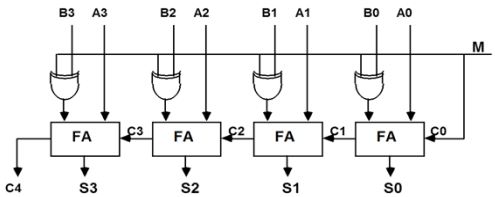
4-bit Binary Incrementer

Arithmetic Micro operations

- The basic arithmetic micro operations are
  - Addition R3 ←R1+R2
  - Subtraction R3 ←R1-R2 or R3 ←R1+R2+1
  - Increment R2 ←R2+1
  - Decrement R2 ←R2-1

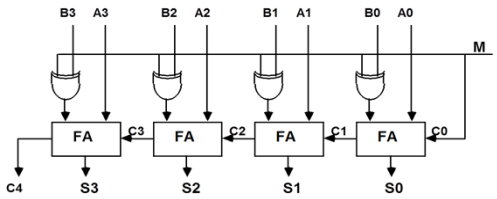
- The additional arithmetic micro operations are
  - Add with carry
  - Subtract with borrow
  - Transfer/Load

Binary Adder Subtractor



- Mode input M controls the operation
  - M=0 ---- adder
  - M=1 ---- Subtractor

Binary Adder Subtractor



- Mode input M controls the operation
  - M=0 ---- adder
  - M=1 ---- Subtractor

Logical Micro operations

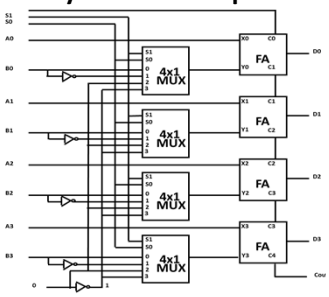
X	Y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

TABLE 4-5. Truth Table for 16 Functions of Two Variables

Boolean function	Microoperation	Name	Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear	$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_1 = xy$	$F \leftarrow A \wedge B$	AND	$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex-NOR
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$		$F_{10} = y'$	$F \leftarrow \overline{B}$	Compl-B
$F_3 = x$	$F \leftarrow A$	Transfer A	$F_{11} = x+y'$	$F \leftarrow A \vee \overline{B}$	
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$		$F_{12} = x'$	$F \leftarrow \overline{A}$	Compl-A
$F_5 = y$	$F \leftarrow B$	Transfer B	$F_{13} = x'+y$	$F \leftarrow \overline{A} \vee B$	
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Ex-OR	$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_7 = x+y$	$F \leftarrow A \vee B$	OR	$F_{15} = 1$	$F \leftarrow$ all 1's	set to all 1's

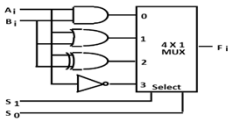
TABLE 4-6. Sixteen Logic Microoperations

Binary Arithmetic Operations



Select S_1	S_0	C_{in}	Input Y	Output $D = A + Y + C_{in}$	Micro operation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with carry
0	1	0	$\overline{B}$	D = A + $\overline{B}$	Subt. with borrow
0	1	1	$\overline{B}$	D = A + $\overline{B}$ + 1	Subtract
1	0	0	0	D = A	Transfer A
1	0	1	0	D = A + 1	Increment A
1	1	0	1	D = A - 1	Decrement A
1	1	1	1	D = A	Transfer A

Hardware Implementation Of Logical Operations



Function table

S_1	S_0	Output	μ-operation
0	0	$F = A \wedge B$	AND
0	1	$F = A \vee B$	OR
1	0	$F = A \oplus B$	XOR
1	1	$F = A'$	Complement

Logical Micro operations

- Logic microoperation
  - Logic microoperations consider *each bit of the register separately* and treat them as binary variables
    - » *exam)* $P : R1 \leftarrow R1 \oplus R2$

      1010 Content of R1
      + 1100 Content of R2
      0110 Content of R1 after P=1
  - Special Symbols
    - » Special symbols will be adopted for the logic microoperations *OR(∨), AND(∧)*, and *complement(a bar on top)*, to distinguish them from the corresponding symbols used to express Boolean functions
    - » *exam)* $P + Q : R1 \leftarrow R2 + R3,\ R4 \leftarrow R5 \vee R6$

      Logic OR Arithmetic ADD
- List of Logic Microoperation
  - Truth Table for 16 functions for 2 variables : *Tab. 4-5*
  - 16 Logic Microoperation : *Tab. 4-6*
- Hardware Implementation
  - 16 microoperation → Use only 4(AND, OR, XOR, Complement)
  - One stage of logic circuit

  ☆All other Operation can be derived

Applications of Logic Micro operations

- Logic micro operations can be used to manipulate individual bits or a portions of a word in a register
- Consider the data in a register A. In another register, B, is bit data that will be used to modify the contents of A

➢Selective-set	$A \leftarrow A + B$
➢Selective-complement	$A \leftarrow A \oplus B$
➢Selective-clear	$A \leftarrow A \bullet B'$
➢Mask (Delete)	$A \leftarrow A \bullet B$
➢Clear	$A \leftarrow A \oplus B$
➢Insert	$A \leftarrow (A \bullet B) + C$
➢Compare	$A \leftarrow A \oplus B$

Applications of Logic Micro operations

1. In a **selective set operation**, the bit pattern in B is used to set certain bits in A

1 1 0 0 A_t
1 0 1 0 B
1 1 1 0 A_{t+1} (A ← A + B)

If a bit in B is set to 1, that same position in A gets set to 1, otherwise that bit in A keeps its previous value.

2. In a **selective complement operation**, the bit pattern in B is used to complement certain bits in A

1 1 0 0 A_t
1 0 1 0 B
0 1 1 0 A_{t+1} (A ← A ⊕ B)

If a bit in B is set to 1, that same position in A gets complemented from its original value, otherwise it is unchanged.

Applications of Logic Micro operations

3. In a **selective clear operation**, the bit pattern in B is used to *clear* certain bits in A

1 1 0 0 A_t
1 0 1 0 B
0 1 0 0 A_{t+1} (A ← A · B')

If a bit in B is set to 1, that same position in A gets set to 0, otherwise it is unchanged

4. In a **mask operation**, the bit pattern in B is used to *clear* certain bits in A

1 1 0 0 A_t
1 0 1 0 B
1 0 0 0 A_{t+1} (A ← A · B)

If a bit in B is set to 0, that same position in A gets set to 0, otherwise it is unchanged

Applications of Logic Micro operations

5. In a **clear operation**, if the bits in the same position in A and B are the same, they are cleared in A, otherwise they are set in A

1 1 0 0 A_t
1 0 1 0 B
0 1 1 0 A_{t+1} (A ← A ⊕ B)

Applications of Logic Micro operations

6. An **insert operation** is used to introduce a specific bit pattern into A register, leaving the other bit positions unchanged

This is done as

–A mask operation to clear the desired bit positions, followed by

–An OR operation to introduce the new bits into the desired positions

–Example: Suppose you wanted to introduce 1010 into the low order four bits of A:

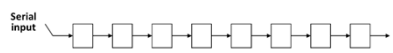
1101 1000 1011 0001 A (Original)
1101 1000 1011 1010 A (Desired)

```
1101 1000 1011 0001      A (Original)
1111 1111 1111 0000      Mask
1101 1000 1011 0000      A (Intermediate)
0000 0000 0000 1010      Added bits
1101 1000 1011 1010      A (Desired)
```

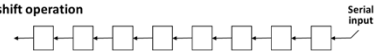
Shift Micro operations

- **There are three types of shifts**
  - ***Logical shift***
  - ***Circular shift***
  - ***Arithmetic shift***
- **What differentiates them is the information that goes into the serial input**

- **A right shift operation**



- **A left shift operation**



Shift Micro operations

•In a Register Transfer Language, the following notation is used

shl for a logical shift left

shr for a logical shift right

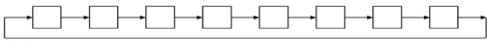
Examples:

•R2 ← *shr* R2

•R3 ← *shl* R3

Shift Micro operations

- In a circular shift the serial input is the bit that is shifted out of the other end of the register.
- A right circular shift operation:



- A left circular shift operation:

- In a RTL, the following notation is used
  - *cil* for a circular shift left
  - *cir* for a circular shift right
  - Examples:
    - R2 ← *cir* R2
    - R3 ← *cil* R3

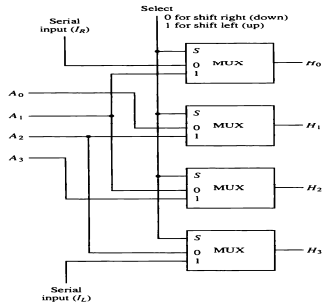
Shift Micro operations RTL

Symbolic designation	Description
$R \leftarrow$ shl R	Shift-left register R
$R \leftarrow$ shr R	Shift-right register R
$R \leftarrow$ cil R	Circular shift-left register R
$R \leftarrow$ cir R	Circular shift-right register R
$R \leftarrow$ ashl R	Arithmetic shift-left R
$R \leftarrow$ ashr R	Arithmetic shift-right R

Shift Micro operations

- An arithmetic shift is meant for signed binary numbers (integer)
- An arithmetic left shift multiplies a signed number by two
- An arithmetic right shift divides a signed number by two
- Sign bit : 0 for positive and 1 for negative
- The main distinction of an arithmetic shift is that it must keep the sign of the number the same as it performs the multiplication or division
- A right arithmetic shift operation
- A left arithmetic shift operation

Hard ware for shift operations



Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

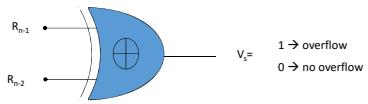
Shift Micro operations

- Example: Assume R1=11001110, then:
  - Arithmetic shift right once : R1 = 11100111
  - Arithmetic shift right twice : R1 = 11110011
  - Arithmetic shift left once : R1 = 10011100
  - Arithmetic shift left twice : R1 = 00111000
  - Logical shift right once : R1 = 01100111
  - Logical shift left once : R1 = 10011100
  - Circular shift right once : R1 = 01100111
  - Circular shift left once : R1 = 10011101

Overflow handling in shift operations

- An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow

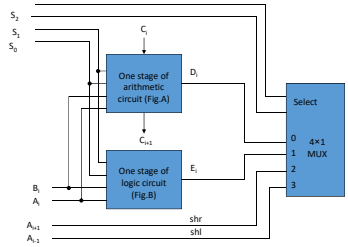
$$V_s = R_{n-1} \oplus R_{n-2}$$



Overflow Condition of Arithmetic Shift Registers

arithmetic shift An *arithmetic* shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2. The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form. Figure 4-11 shows a typical register of n bits. Bit R_{n-1} in the leftmost position holds the sign bit. R_{n-2} is the most significant bit of the number and R_0 is the least significant bit. The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right. Thus R_{n-1} remains the same, R_{n-2} receives the bit from R_{n-1}, and so on for the other bits in the register. The bit in R_0 is lost.

Arithmetic Logic Shift Unit



Overflow Condition of Arithmetic Shift Registers

The arithmetic shift-left inserts a 0 into R_0, and shifts all other bits to the left. The initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2}. A sign reversal occurs if the bit in R_{n-1} changes in value after the shift. This happens if the multiplication by 2 causes an overflow. An overflow occurs after an arithmetic shift left if initially, before the shift, R_{n-1} is not equal to R_{n-2}. An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow.

$$V_s = R_{n-1} \oplus R_{n-2}$$

If $V_s = 0$, there is no overflow, but if $V_s = 1$, there is an overflow and a sign reversal after the shift. V_s must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

Arithmetic Logic Shift Unit Function Table

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F

Arithmetic Logic Shift Unit

Instead of having individual registers performing the micro-operations directly, computer systems employ a number of storage registers connected to a common operational unit called an Arithmetic Logic Unit **(ALU).**